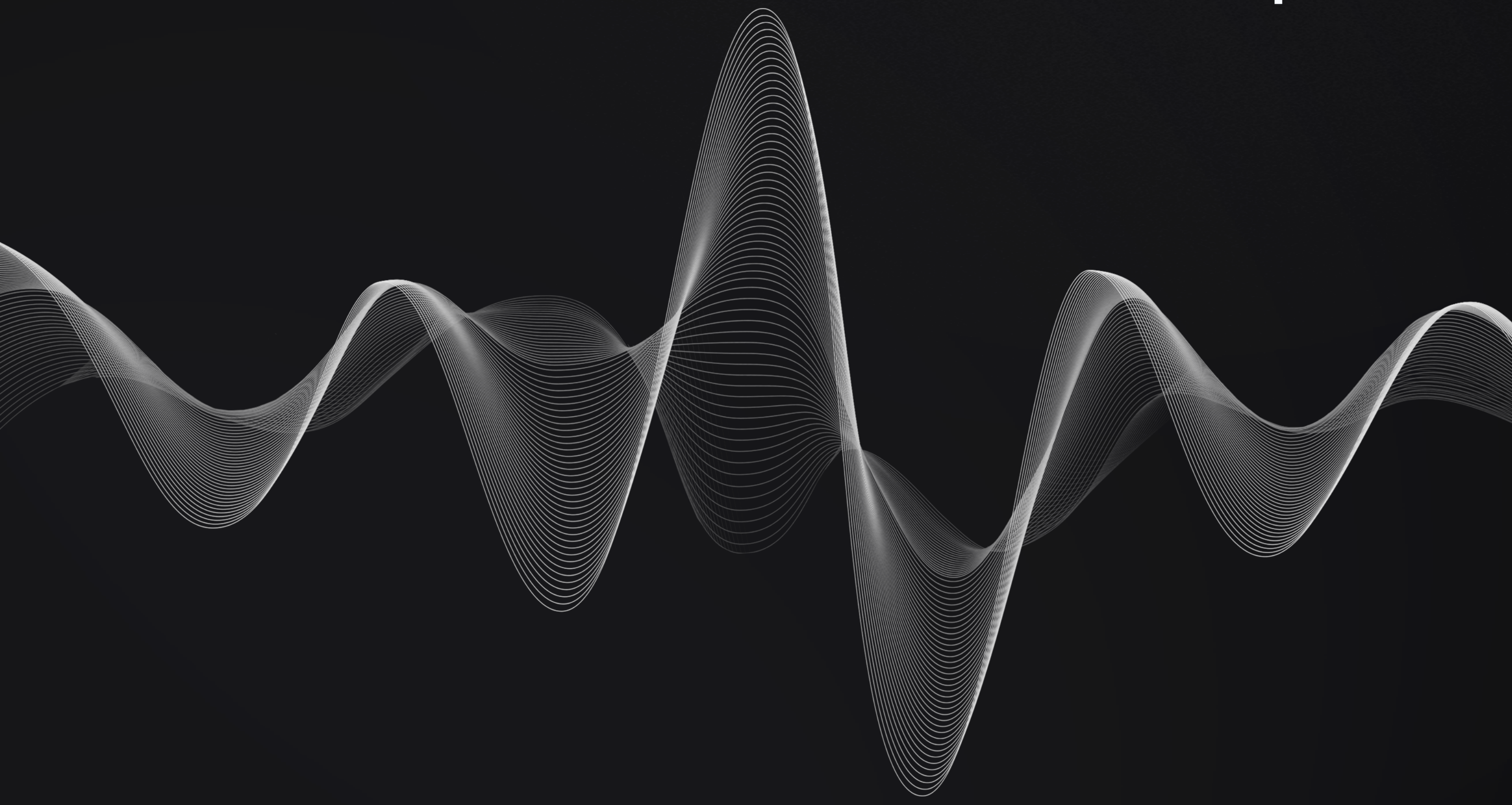
RESONANCE

Amulet Finance

Smart Contracts Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.2)

Audit_Report_AMLT-PRO_FINAL_22

Jun 26, 2024	○	v0.1	Michal Bazyli: Initial draft
Jun 26, 2024	○	v0.2	Michal Bajor: Added findings
Jun 27, 2024	🟢	v1.0	Charles Dray: Approved
Jul 1, 2024	○	v1.1	Michal Bazyli: Reviewed findings
Jul 1, 2024	🟡	v2.0	Charles Dray: Finalized
Jul 3, 2024	○	v2.1	Michal Bazyli: Reviewed finding
Jul 3, 2024	○	v2.2	Charles Dray: Published

Points of Contact	Robyn	Chambers	robyn@gaiapx.com
	Charles Dray	Resonance	charles@resonance.security
Testing Team	Michał Bazyli	Resonance	michal.bazyli@resonance.security
	João Simões	Resonance	joao.simoes@resonance.security
	Ilan Abitbol	Resonance	ilan.abitbol@resonance.security
	Michal Bajor	Resonance	michal.bajor@resonance.security

Copyright and Disclaimer

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

Contents

Executive Summary

The team building **Amulet Finance** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between June 12, 2024 and June 27, 2024. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 10 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Amulet Finance with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.Y

System Overview

Amulet Finance is a decentralized finance (DeFi) platform that provides self-repaying loans on staked assets within the Cosmos ecosystem. Users can deposit tokens to receive an advance on their staking rewards. These advances automatically repay themselves over time using the rewards, avoiding liquidations since the loans are denominated in matching synthetic assets.

Repository Coverage and Quality

Code	Tests	Documentation
N/A	N/A	N/A

This section of the report has been concealed by the request of the customer.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: `Amulet-Dev/contracts/contracts`
- Hash: 6cdaf633dee532832027c432a450ac244184788e
- Files:

- `contracts/mint/contract.rs`
- `contracts/vault/generic-lst/strategy.rs`
- `contracts/vault/remote-pos/strategy.rs`
- `crates/core/mint.rs`
- `crates/core/vault.rs`
- `crates/cosmwasm/mint.rs`
- `crates/cosmwasm/strategy.rs`
- `crates/cosmwasm/vault.rs`
- `crates/neutron/token_factory.rs`
- `crates/num/lib.rs`
- `crates/pos-reconcile-fsm/lib.rs`
- `crates/pos-reconcile-fsm/types.rs`

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust CosmWasm

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust CosmWasm audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Frontrunning attacks
- Unsafe third party integrations
- Denial of service
- Access control issues
- Inaccurate business logic implementations
- Incorrect gas usage

- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ▂▂▂▃▂▂▂ **"Quick Win"** Requires little work for a high impact on risk reduction.
- ▂▃▂▅▆▃▂ **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ▂▅▂▇█▅▂ **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Inflexibility in Whitelist Management	▂▃▂▅▆▃▂	Resolved
RES-02	Potential Denial of Service in Active Unbondings Query	▂▂▂▃▂▂▂	Resolved
RES-03	Potential Denial of Service in All Assets Query	▂▃▂▅▆▃▂	Resolved
RES-04	Usage of Vulnerable Packages	▂▃▂▅▆▃▂	Acknowledged
RES-05	Lack of Event Emission in the Codebase	▂▃▂▅▆▃▂	Acknowledged

Inflexibility in Whitelist Management

Info **RES-AMLT-PRO01** Business Logic **Resolved**

Code Section

- `crates/cosmwasm/mint.rs`

Description

In the `handle_cmd` function, the `ConfigCmd::Whitelist` command allows for setting a minter as whitelisted by storing a boolean flag (`enabled`) in the storage. However, the current implementation does not provide a mechanism to completely remove a minter from the whitelist storage. This can lead to unnecessary storage usage, as entries for minters that are no longer needed will still occupy space.

Recommendation

Implement functionality to completely remove a minter from the whitelist storage when it is no longer needed. This can be achieved by adding a conditional check to remove the storage entry when `enabled` is `false`.

Status

The issue has been fixed in a36a65e31727b062e126aa78f031e82edfd8ffe8.

Potential Denial of Service in Active Unbondings Query

Info **RES-AMLT-PRO02** Denial of Service **Resolved**

Code Section

Not specified

Description

The `all_active_unbondings` function is used in the `QueryMsg::ActiveUnbondings` query to retrieve all active unbonding records. This function does not implement pagination, which can lead to performance issues and potential denial of service (DoS) attacks, especially when the number of unbonding records is large.

Recommendation

Implement pagination in the `all_active_unbondings` function and update the `QueryMsg::ActiveUnbondings` query to handle pagination parameters. This will ensure that only a manageable subset of records is retrieved at a time, preventing performance issues and mitigating the risk of DoS.

Status

The issue has been fixed in 1aa01bc71eff94ee089c3917f95fc3dd5f59db5b.

Potential Denial of Service in All Assets Query

Info **RES-AMLT-PRO03** Denial of Service **Resolved**

Code Section

- crates/cosmwasm/mint.rs#L177

Description

The `AllAssets` query lacks mechanisms to control the volume of data it retrieves, potentially leading to a denial of service (DoS) if the dataset becomes excessively large. This unrestricted data retrieval could strain system resources, leading to performance bottlenecks or system crashes.

Recommendation

Introduce pagination or a limit on the number of assets returned in one query to manage the load effectively and prevent potential DoS attacks. Implementing these controls will help maintain the system's reliability and responsiveness.

Status

The issue has been fixed in 3b387a858776d9478bb89310bdc8e1e64ecf8cad.

Usage of Vulnerable Packages

Info **RES-AMLT-PRO04** Public Vulnerabilities **Acknowledged**

Code Section

Not Specified

Description

The `cargo audit` scan has identified a security vulnerability in the `curve25519-dalek` crate, version 3.2.0. The specific issue is a timing variability in the `Scalar29::sub` and `Scalar52::sub` functions, which can potentially lead to side-channel attacks. This vulnerability has been documented under the advisory ID RUSTSEC-2024-0344.

Recommendation

Upgrade the `curve25519-dalek` crate to version 4.1.3 or later, as this version includes fixes for the timing variability issue. Ensure that all dependent crates are also updated accordingly to avoid compatibility issues.

Status

> *The issue was acknowledged by Amulet Finance team. The development team stated "The flagged transitory dependency is from the direct cosmwasm-std@1.5.5 dependency, which is set to the highest version supported by the Neutron network. However, none of the contracts use the cryptographic APIs provided by cosmwasm-std and are not affected by the CVE."*

Lack of Event Emission in the Codebase

Info **RES-AMLT-PRO05** Business Logic **Acknowledged**

Code Section

Not Specified

Description

The current codebase lacks the use of event emission to log significant actions and state changes. Events are a critical component in blockchain systems, providing transparency and traceability by recording important actions on the blockchain. Without event emission, it becomes challenging to track and audit the system's behavior, making debugging and monitoring difficult.

Recommendation

Implement event emission throughout the codebase to log significant actions and state changes. This will enhance the transparency, auditability, and traceability of the contract. Events should be emitted for actions such as the creation and removal of synthetic assets, whitelist updates, and unbonding activities.

Status

> *The issue was acknowledged by Amulet Finance team. The development team stated " Events will be added to ExecuteMsg responses before mainnet deployment once everything else is finalised."*

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.